Computer Science Dept. / 2nd year 1st Course (2024-2025)
Microprocessor
Dr.Haleema Essa

❖ 8086 CPU Architecture, *Continue…*

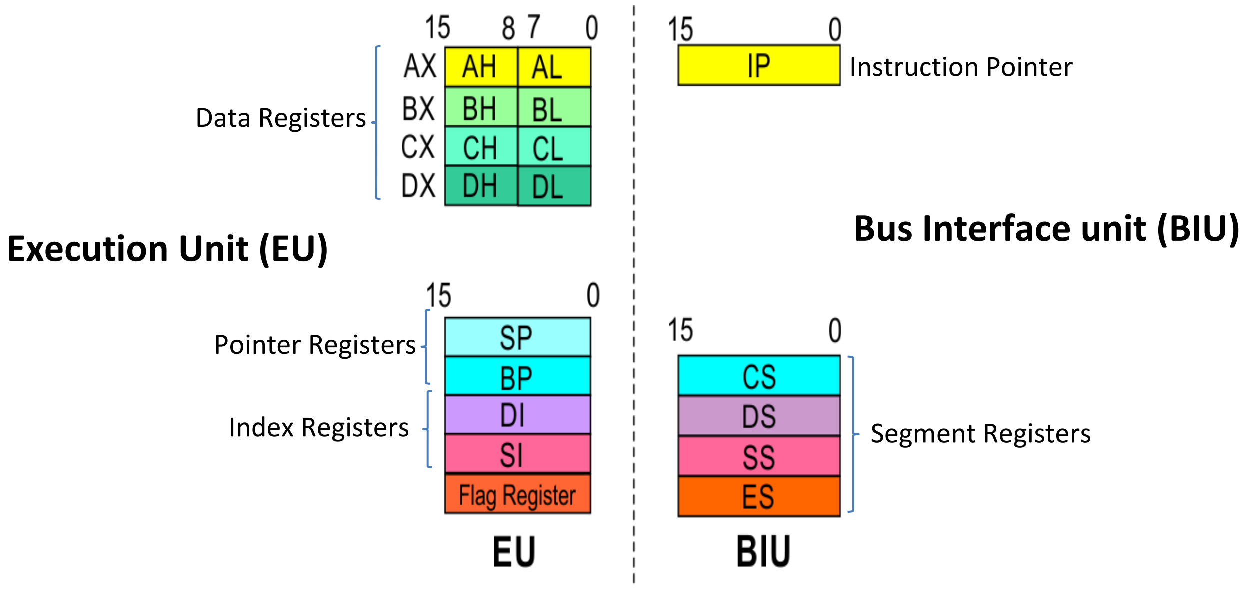
8086 Microprocessor does not have a RAM or ROM inside it. However, it has internal registers for storing intermediate and final results and interfaces with memory located outside it through the System Bus.

The size of the internal registers (present within the chip) indicates how much information the processor can operate on at a time.

The 8086 CPU is organized as ***two separate units***, called the Bus Interface Unit (***BIU***) and the Execution Unit (***EU***).

- ✓ The ***BIU*** provides *H/W functions*, including generation of the memory and I/O addresses for the transfer of data between outside the CPU and the EU.
- ✓ The ***EU*** receives program instruction codes and data from the BIU, executes these instructions, and store the results in the general registers. By passing the data back to the BIU, data can also be stored in a memory location or written to an output device.

Note that the EU has no connection to the system buses. It receives and outputs all its data through the ***BIU***. The basic architecture of 8086 is shown below.



The Two Sections of the 8086 Microprocessor

❖ The Main Components Execution Unit (EU):

The main components of the EU are:

- Control Circuit.
- Instruction Register and Instruction Decoder.
- ALU.
- General Purpose Registers.
- Special Purpose Registers.
- Flag/Status Register.

EU unit performs the following functions:

- It Fetches instructions from the Stack in BIU, and decodes it.
- It performs the logic & arithmetic operation on memory or register, using the ALU.
- It stores the information temporary in the register array.
- Sends request signals to the BIU to access the external module.

Register organization of 8086 Microprocessor

❖ A General purpose register:

Following is the list of some of the most common registers used in a basic computer:

A. **Data Registers**
B. **Pointer Registers**
C. **Index Registers**

A. Data Registers include:

I. Accumulator Register (AX):

Consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte.

II. Base Register (BX):

Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. This is the only general purpose register whose contents can be used for addressing the 8086 memory. *All memory references utilizing this register content for addressing use* ***DS*** *as the default segment register.*

III. Counter Register (CX):

Consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. *Instructions such as SHIFT, ROTATE and LOOP use the contents of CX as a counter.*

IV. Data Register (DX):

Consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Used to hold the data for multiplication or division.

B- Pointer and Index Registers:

The 8086 has four other general-purpose registers, *two pointer registers **SP and BP**, and two index registers **DI and SI***. These are used to store what is called offset addresses. These registers are unlike the general-purpose data registers, the pointer and index registers are ***only accessed as words (16 bits).***

Pointer Registers include:

I. SP (Stack Pointer):

Points to Stack top. Stack is in Stack Segment, used during instructions like PUSH, POP, CALL, RET etc. It is *used with **SS*** to access the stack segment.

II. BP (Base Pointer):

This is base pointer register *pointing to **data** in Stack Segment*. Unlike SP, we can use BP to access data ***in the other segments***. BP can hold offset address of any location in the stack segment. It is used to access *random locations* of the stack.

C- Index Registers include:

These types of registers useful for doing vector/array operations. These also used to reduce the amount of memory used and increased execution speed.

I. SI (Source Index):

This is source index register, which is used *to point to memory locations in the Data Segment addressed by* ***DS***. Thus, when we increment the contents of SI, we can easily access consecutive memory locations. It holds offset address in *Data* Segment during string operations.

II. DI (Destination Index):

This is destination index register performs the same function as SI but in *Extra Segment* ***ES*** not in Data Segment DS. It holds offset address in *Extra* Segment during string operations.

❖ A special purpose register:

Some registers serve specific functions within the CPU. Several of the more important of these registers are ***Instruction Register, Program Counter (Instruction Pointer), and Status/Flag Register***.

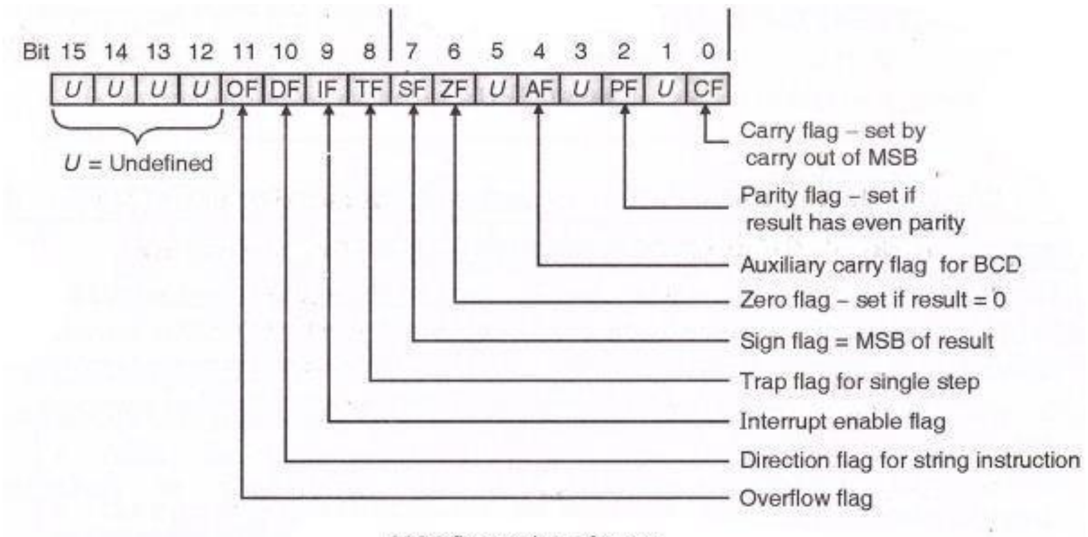
Status / Control Flag Register of 8086

The Flag register is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0).

The flag register is a part of ***EU*** (Execution Unit). It is a 16-bit register with each bit corresponding to a flip-flop. It indicates some condition produced by the execution of an instruction. For example, the zero flag (ZF) will set if the result of execution of an instruction is zero.

The Flag register is the status register in Intel x86 microprocessors that contains the current state of the processor. Its successors, the EFLAGS and RFLAGS registers, are 32 bits and 64 bits wide, respectively. The wider registers retain compatibility with their smaller predecessors.

Flags register consists of ***9 active*** *flags out of 16*. The remaining **7** flags marked '**U**' are undefined flags. These **9** flags are of two types: ***6*** *Status flags &* ***3*** *Control flags*. So, you can divide the flag bits into two sections. *The Status Flags, and the Control Flags*.


Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
U U U U OF DF IF TF SF ZF U AF U PF U CF
U = Undefined
Carry flag – set by carry out of MSB
Parity flag – set if result has even parity
Auxiliary carry flag for BCD
Zero flag – set if result = 0
Sign flag = MSB of result
Trap flag for single step
Interrupt enable flag
Direction flag for string instruction
Overflow flag

8086 flag register format

A. Status flags:

1. ***Carry flag (CF).***
2. ***Parity flag (PF).***
3. ***Auxiliary carry flag (AF).***
4. ***Zero flag (ZF).***
5. ***Sign flag (SF).***
6. ***Overflow flag (OF).***

B. Control flags:

1. Trap flag (TF):

It is used for ***single step control***. It allows user to execute one instruction of a program at a time for debugging. When the trap flag is set, the program can be run in single step mode.

2. Interrupt enable flag (IF):

It is used to mask (disable) or unmask (enable) the INTR interrupt. If a user sets the IF flag, the CPU will recognize external interrupt requests. It determines whether or not the CPU will respond to mask-able interruptions.

- *If the flag is set to 1, mask-able interrupts will be handled.*
- *If cleared (set to 0), such interrupts will be ignored.*

3. Direction flag (DF):

It is used with ***string operations***. When set (1), it causes the string instructions to process strings *from right to left* .otherwise the string is process from *left to right*.

- ✓ *DF = 0 (Auto Increment)*
- ✓ *DF = 1 (Auto Decrement)*

❖ *Process Control Instructions in 8086:*

The Process Control Instructions in 8086 are:

- ❖ ***STC: ST*** *means* ***Set*** *(make it 1),* ***C*** *means Carry flag.*
- ❖ ***CLC: CL*** *means* ***Clear*** *(make it 0),* ***C*** *means Carry flag.*
- ❖ ***CMC: CM*** *means* ***Complement*** *(Inverse the value).*
- ❖ ***STD: Set D****irection flag.*
- ❖ ***CLD: Clear D****irection flag.*
- ❖ ***STI: Set I****nterrupt flag.*
- ❖ ***CLI: Clear I****nterrupt flag.*

Below is the explanation of each type of these instructions:

❖ ***STC Instruction:***

This instruction ***sets*** *the carry flag, STC and all of the following instructions* ***does not affect*** *any other flag.*

❖ ***CLC Instruction:***

This instruction resets the carry flag to ***zero****.*

❖ ***CMC Instruction:***

This instruction complements the carry flag.

❖ ***STD Instruction:***

This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instructions.

❖ ***CLD Instruction:***

This instruction is used to reset the direction flag to zero, so that SI and/or DI can be incremented automatically after execution of string instructions.

❖ ***STI Instruction:***

This Process Control Instructions in 8086 sets the interrupt flag to one. This enables INTR interrupt of the 8086.

❖ ***CLI Instruction:***

This instruction resets the interrupt flag to zero. Due to this Process Control Instructions in 8086 will not respond to an interrupt signal on its INTR input.

8086 External Hardware Synchronization Instructions:

The 8086 External Hardware Synchronization Instructions are:

- ***HLT***
- ***WAIT***
- ***ESC***
- ***LOCK***
- ***NOP***

✓ ***HLT Instruction:***

*The HLT instruction will cause the 8086 to **stop fetching and executing instructions**. The 8086 will enter a halt state. The only ways to get the processor out of the halt state are with an interrupt signal on the **INTR** pin, an interrupt signal on the **NMI** pin, or a reset signal on the **RESET** input.*

- ***WAIT Instruction:***

*When this instruction executes, the 8086 enters an idle condition where it is doing no processing. The 8086 will stay in this idle state until a signal is asserted on the 8086 TEST input pin, **or** until a valid interrupt signal is received on the INTR or the NMI interrupt input pins.*

- ***ESC Instruction:***

*This instruction is used to **pass instructions to a coprocessor** such as the 8087-math coprocessor which shares the address and data bus with an 8086. Instructions for the coprocessor are represented by a **6-bit code** embedded in the escape instruction.*

*When the 8086 fetches an ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6-bit code specified in the instruction. In most cases the 8086 treats the ESC instruction as a **NOP**. In some cases, the 8086 will access a data item in memory for the coprocessor.*

- ***LOCK Instruction:***

*The LOCK prefix allows a microprocessor to make sure that another processor does not take control of the system bus while it is in the middle of **a critical instruction**, which uses the system bus. The LOCK prefix is put in front of the critical instruction.*

- ***NOP Instruction:***

*At the time of execution of NOP instruction**, no operation is performed except fetch and decode**. It takes three clock cycles to execute. NOP instruction does not affect any flag. This instruction is used to fill in time delays or to delete and insert instructions in the program while trouble shooting.*